2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics

AIM 2005

24—28 July 2005

Monterey, California
U. S. A.

PROGRAM

http://www.aim2005.org

IEEE

Co-sponsored by
IEEE Industrial Electronics Society
IEEE Robotics and Automation Society
ASME Dynamic Systems and Control Division

Proceedings of the 2005 IEEE/ASME
International Conference on Advanced Intelligent Mechatronics
Monterey, California, USA, 24-28 July, 2005

Composite Component Framework for RT-Middleware (Robot Technology Middleware)

Noriaki Ando, Takashi Suehiro, Kosei Kitagaki, Tetsuo Kotoku and Woo-Keun Yoon
Intelligent Systems Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2,Tsukuba,Ibaraki 305-8568, Japan
{*n-ando, t.suehiro, k.kitagaki, t.kotoku, wk.yoon*}*@aist.go.jp*

***Abstract*— We have studied a framework of RT-Component which promotes application of Robot Technology (RT) in various field. In this paper, we will discuss robotic system development methodology and our RT-Middleware concepts. The system development methodology using RT-Component, and new framework to make composite component for RT-Component will be shown. A evaluation of composite component framework, which realizes low level and real-time composition of independent RT-Components, will be derived. Finally conclusion and future work will be described.**

***Index Terms*— RT (Robot Technology), software component, middleware, robot system, system integration.**

I. Introduction

In recent years, studies for the robot system integration developing complex robotic systems by integrating basic robot functions are becoming important in robotics. A lot of robotic basic functions have been studied and developed enough to realize simple intelligent tasks which makes human daily life more convenient. Robotics research for full-scale application integrating robotic functional elements is also active now. Intelligent environment and ubiquitous computing are potential examples of full-scale application for robotic system integration.

From such backgrounds, the necessity of the systematic knowledge for robot system integration has increased. As shown in a Figure 1, the methodology of robot system integration independent from persons' experience and knowhow, and robot system platform to support it is also needed.

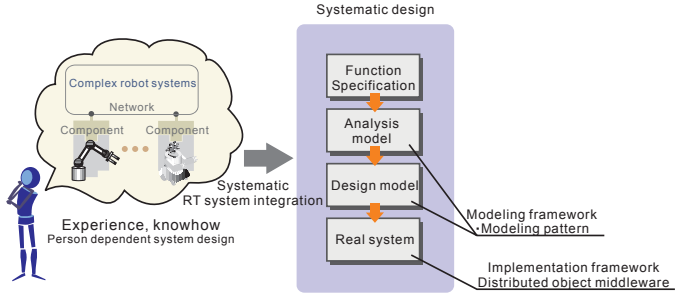

Fig. 1. A Robot Systems Modeling Flow: The RT system should be modeled and designed through systematic design flow independent from the persons' experience and knowhow.

We started RT-Middleware project from 2002 under NEDO's (New Energy and Industrial Technology Development Organization) "Robot challenge program". Basic functions for robot software platform, which supports complex robot system integration, have been studied. The purpose of this project is to establish basic infrastructure for robot system integration, which make it possible to develop systems with new functions by modularized software components.

Research of software modularisation of robot functions and development of software libraries for robot system integration are performed actively in recent years. **ORiN** (Open Resource interface for the Network/Open Robot interface for the Network) is middleware, which offers the standard communication interface over various FA (factory automation) equipment including a robot [1], [2].

Orocos is a free software project that includes a set of class libraries and application framework, and a hard-real-time kernel for all possible feedback control applications [3], [4].

ORCA (Open Robot Controller architecture) developed in Toshiba is HORB (Java ORB developed in AIST) based robot controller architecture [5], [6]. SONY is actively promoting **OPEN-R**, which is the standard interface and platform for the entertainment robot system [7].

On the other hand, one of our purpose is to define standard interface of the software component for robots, which makes interconnection possible. Since the standard interface specification is free and open, any vendors can implement middleware based on this interface specification. To provide open-source middleware based on those interface specifications is another important purpose. Getting feedback from actual research use and application use, it is expected that the improvement of the interface specification will be advanced. Final target is to establish a systematic robot system design theory derived from the knowledge of component-oriented robot system integration. If robot systems with new functions can be constructed more flexibly, it can satisfy every users' needs individually, which cannot be satisfied now. Thus, it is expected that the conventional robot industry mainly restricted to the manufacturing field will be expanded to the nonmanufacturing field like support robots for daily life.

In this paper, a RT-Component object model based on distributed object middleware, and new component framework "composite component" will be mentioned. First, an outline about the basic structure of the RT-Component derived from this discussion will also be mentioned. Next, the architecture of the core part of the RT-Component will be shown. The new framework, which realizes component composition and

real-time cooperation, composite component framework, will be introduced. A force-controlled manipulator system, which was implemented using composite component framework, and experiments and its results will be shown and finally a conclusion is described.

II. Component Oriented Development

Robotics research has been mainly focused on analysis of mechanical function, machine intelligence, and realization of simple robot function relatively. On the other hand, the synthesis knowledge for systematic integration of robot systems is also necessary to apply simple robot functions to the real world. It would be said that the present robotics has little knowledge for the synthesis, and that knowledge is not systematized even if it exists.

If learning from a general software development flow, a robot system development flow shown in Figure 2 would occur.

The knowledge for complex robot system development, which is performed by researcher and developer based on their experience and knowhow, can be roughly classified into system analysis knowledge, system design knowledge, system implementation knowledge.

Furthermore, the analysis knowledge is classified into the knowledge to analyze the required specification to adapt technology to the system (analysis pattern), and the framework to realize it (analysis model framework). Moreover, it consists of the knowledge (design pattern) and the framework (design framework), which bring a design into more concrete level for implementation. Our RT-Middleware includes an implementation framework for it.

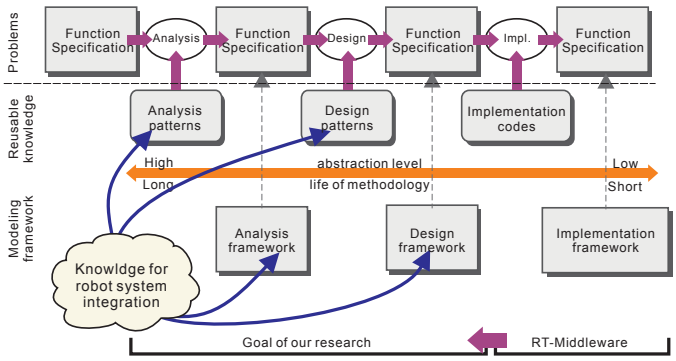

Fig. 2. RT system development flow. This flow is composed of system analysis, system design, system implementation. RT-Middleware supports system implementation level. Deriving systematic system analysis and design methods is our final goal of this research.

Generally, as shown in Figure 2, the abstraction level of the knowledge close to implementation is low and is short-life, and the abstraction level of the knowledge close to analysis phase is high and is long-life.

If generalized analysis patterns and modeling frameworks are obtained, the robot system development can easily follow to technological advancement. Our research aim is to obtain the abstract design patterns and frameworks, and the RT-Middleware is an implementation framework and robot system platform. This is the concept of the RT-Middleware.

III. RT-Component Object Model

We chose CORBA as distributed object middleware for platform independency and programming language independency, and tried modeling of RT module on CORBA. We propose the RT-Component, as a RT module unit model based on the distributed object model.

An RT-Component consists of the following objects and interfaces.

- Component object.
- Activity.
- *InPort* as input port object.
- *OutPort* as output port object.
- Command interfaces.

The general distributed object model can be described as some interfaces that contain operations with parameters and a return value. On the other hand, the RT-Component model has a component object as a main body, activity as a main process unit, input ports (InPort) and output ports (OutPort) as data stream ports.

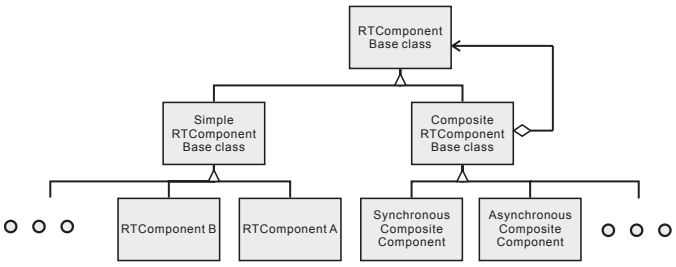

Fig. 3. The RT-Component class hierarchy. User defined component class inherits simple RT-Component base class. The composite component has RT-Component itself by the composite pattern.

Figure 3 shows the hierarchy of RT-Component classes.

Simple RT-Component Base Class: In Figure 3, the simple RT-Component base class is the base class of each new RT-Component class, which is created by the component developer. A component developer can develop his/her component class by inheriting from the RT-Component base class.

Composite RT-Component Base Class: On the RT-Middleware, various granularity RT-Components will be provided by component developer. In this case, such a composite structure or a nested structure are useful for hierarchical robot system integration. To realize the composite structure, the composite pattern is applied to the RT-Component object structure.

The component states and state transition of the component was defined so that various type of RT-Components could be treated as common software parts. By giving a common state transition to RT-Components, and specifying the meaning of states, it is possible to control the action of many components similarly. Various granularity components can be treated similarly, by developing a component according to this state definition. It becomes possible to realize the composite component, which is nested components and grouped components, by defining the component state transition.

A. Activity State of RT-Component

The activity of RT-Component has ten state: `BORN`, `INITIALIZE`, `READY`, `STARTING`, `ACTIVE`, `STOPPING`, `ABORTING`, `ERROR`, `EXITING`, `FATALERROR`, `UNKNOWN`. Figure 4 shows the state transition chart (UML state chart) of RT-Component's activity. RT-Component's methods which are invoked in each state are described in each state block according to the UML notation.

The meaning of method prefixes is the following.

- entry: An atomic action performed on entry to the state.
- do: An iterated action performed while being in the state.
- exit: An atomic action performed on exit from the state.

The state of having only a "entry" method is a transient state, which changes to the next state immediately. The state of having "do" method is a steady state, which can stay at the state. Table I shows the meaning of each RT-Component state.

TABLE I
THE STATES OF THE RT-COMPONENT ACTIVITY. COMPONENT STATES MAKE TRANSITION ACCORDING TO THE STATE TRANSITION CHART SHOWN IN FIGURE 4.

BORN	The Born state of RT-Component. Creating a component instace.
INITIALIZE	The Initializing state. RT-Component initialize process is performed at this state.
READY	The Ready state. This state can be changed to the "Active" state immediately.
STARTING	The Starting state. Just before entering Active state.
ACTIVE	The Active state. A main process is performed at this state.
STOPPING	The Stopping state. A transient state from Active state to Ready state.
ABORTING	The Aborting state. If error has occured at Active state, the state entering here.
ERROR	The Error state. If error has occured, all state will be changed to this state.
EXITING	The Exiting state. Finalize component and changed to the exiting state.
FATALERROR	The Fatal error state. If fatal error has occured all state will be changed to this state.
UNKNOWN	The Unknown state. The state never come here.

A component developer has only to map his/her algorithm or library into each RT-Components state, and he/she can just insert his/her code to the RT-Component framework.

B. InPort/OutPort

In the low level real-time control layer, if a component is considered as the functional unit, which consists of inputs, processing, and outputs, so that it may be exactly expressed with a control block diagram, it will be easy to perform a system configuration.

This input/output model is not so suitable for general usage of the distributed object model. Because the object which sends its data to other objects has to know all objects' complete interface definition. On the other hand, in such low level control layer, data type, number of data and unit of data are more important than interface definition. RT-Component adopted the publisher/subscriber model and defines it as InPort/OutPort.

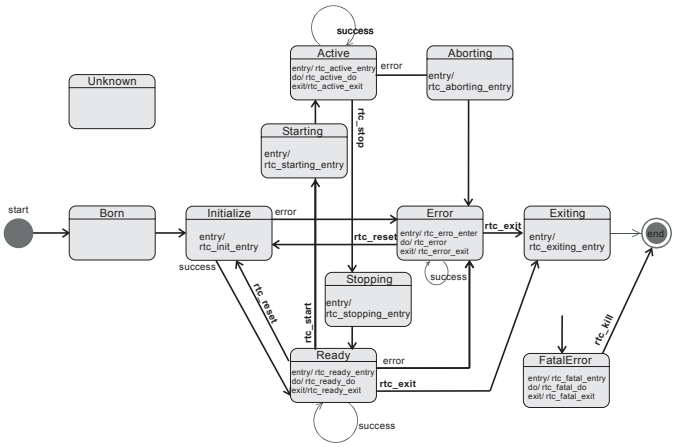

Fig. 4. RT-Component statechart diagram.

1) InPort Object: An input port of RT-Component. InPort receives data from OutPort that calls method of "InPort::put()". This is basic function of InPort.

Other functional InPorts, that raise a signal or invoke a callback method etc., can be implemented as subclasses of the InPort.

2) OutPort Object: An output port of RT-Component. OutPort sends data to InPort that "subscribes" this OutPort, calling "InPort::put()" as "push" type data exchange. "pull" type data exchange calling "OutPort::get()" method is also supported.

OutPort supports some subscription type, "New", "Once", "Periodic", "Periodic New", "New Periodic", "Triggered", "Triggered Priodic", "Periodic Triggered".

For example, the "New" subscription type means that OutPort send data to InPort , which subscribes it, when a new data come from the activity. Due to insufficient space, the detailes of all subscription type cannot be discribed. Other subscription type can also be defined if user needs.

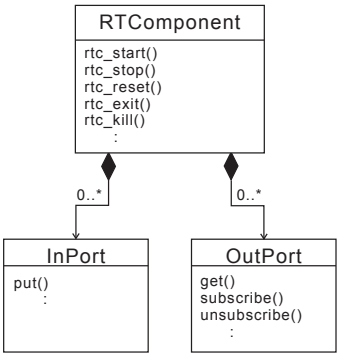

Fig. 5. UML Object Diagram of RT-Component, InPort and OutPort. One RT-Component objects can have 0 and more InPort and OutPort.

As shown in UML Object Diagram of Figure 5, the relation between RT-Component object and Inport/OutPort objects is the composition. An RT-Component object manages object

creation and destruction of InPort and OutPort. Other object or software can ask the RT-Component what kind of InPort/OutPort it has.

IV. Composite Component

A. Problems in Real-time Systems

Until now, based on the above-mentioned component framework, some components have been developed and applied to actual robots. In low level robot control software, they have to be executed in real-time. It was verified that an RT-Component can be a real-time process easily using ARTLINUX. Some real-time RT-Components were developed and they were executed in real-time respectively. However it is not enough that each component is converted to real-time RT-Component. It is not enough just to convert each RT-Component, which constitutes a real-time system, into real-time components respectively.

Now, as shown in Figure 6, we assume that three RT-Components (A, B, C) which constitute a system are mutually dependent on each components' output. RT-Components (A, B, C) are real-time component respectively, and their period time is set to Δt. In the best case (Figure 6), the system which consists of component A, B and C is totally keep the deadline. Generally, if the following task time T less than period time Δt, the real-time task can keep deadline.

$$T = \sum_{i=0}^{n} T_i + \sum_{j=0}^{n-1} d_i < \Delta t \tag{1}$$

Here, T_i means the task time period of i-th component and d_i means the data transfer time period between i-th component and $(i+1)$-th component.

In the worst case in figure 6, the system takes

$$T_{ABC} = 2\Delta t + \sum_{i=A,B,C} T_i + \sum_{j=AB,BC} d_j \tag{2}$$

for completion of a periodic task.

Considering a system to which n components were connected in series, after the first component receives data before the last component outputs data, it takes

$$T_{worst} = (n-1)\Delta t + T. \tag{3}$$

T is minimum task time of equation(1).

For example, assume that those three RT-Components (A, B and C) are a manipulator component, controller component and force sensor component respectively and force-controlled manipulator system is realized by using these components. Each component is executed in real-time as a 1 ms periodic task. In this case, after getting the force value, it takes $2 + t_{manipulator}$ ms to send reference value to manipulator, in the worst case. It is clear that the stability of the system cannot be guaranteed because cycle time is not controllable and depends on the component execution timing.

B. Asynchronous/Synchronous Composite Component

To solve above mentioned problem, new framework for composite component was implemented in RT-Middleware.

The composite components are roughly divided into "the asynchronous composite component" and "the synchronous composite component". The composite components have the following features,

- A composite component can include components to manage them.
- Internal components' InPorts/OutPorts are delegated to the composite component.
- A composite component manages activity states of internal components.

Moreover, the synchronous composite component has the following features,

- Activity states of internal components are completely synchronized.
- Activities of internal components are performed serially in preconfigured order.
- If a thread that invokes each internal component's activity is running in real-time mode, and the response time boundary of method invocation is given and is finite, internal components can be a real-time control task.

The basic asynchronous composite component has the following features,

- States of an internal components do not necessarily have a synchronization.
- Activities of internal components are performed in parallel.

Some types of the asynchronous composite component are possible by the state transition handling type between internal components and the composite component.

An asynchronous composite component, which includes some children components, is handled as one RT-Component. As shown in figure 2, if these components are running in multiple CPUs or multiple machines, these component processes are executed in parallel and tasks are performed efficiently. If these composite components have to be executed as a real-time task, however, the above mentioned problem appears.

On the other hand, a synchronous composite component executes child component activity processes in series and synchronously. This means some component's activities will be executed in a same thread, as if children components activity codes are written in a monolithic code. If a synchronous composite component is executed in a CPU as a real-time process/thread, child component processes are executed as if they are real-time monolithic process/thread.

V. Evaluation and Experiment

A. Component Call Overhead

Evaluation of components' activity method call overhead in synchronous composite components was performed(Figure 7). Some loadable components are loaded into the synchronous composite component and method call time was measured. Experiment was performed on Pentium4 2.8GHz PC with ARTLINUX. Child components were executed in real-time as a 1 ms periodic task.

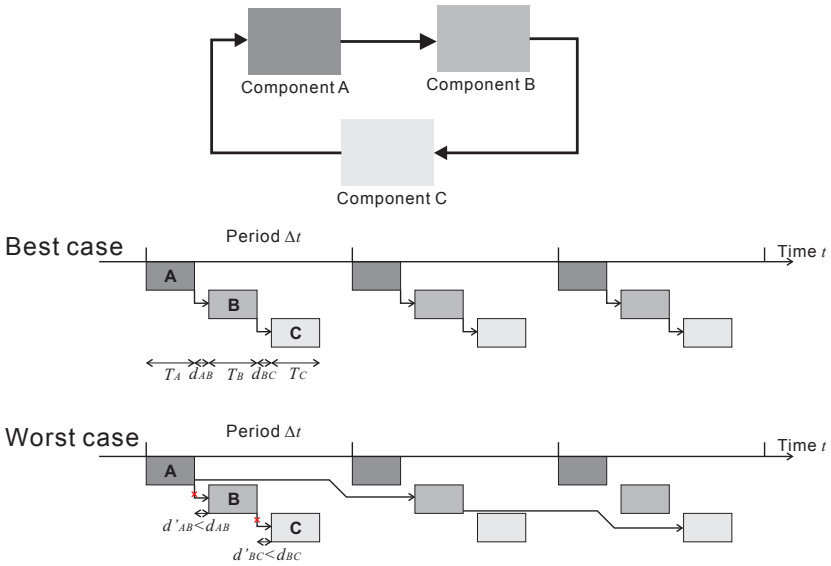

Fig. 6. Combination of independent real-time RT-Component. Each RT-Component running in real-time. Activity execution timing is uncontrollable, because their activities are invoked independently each other.

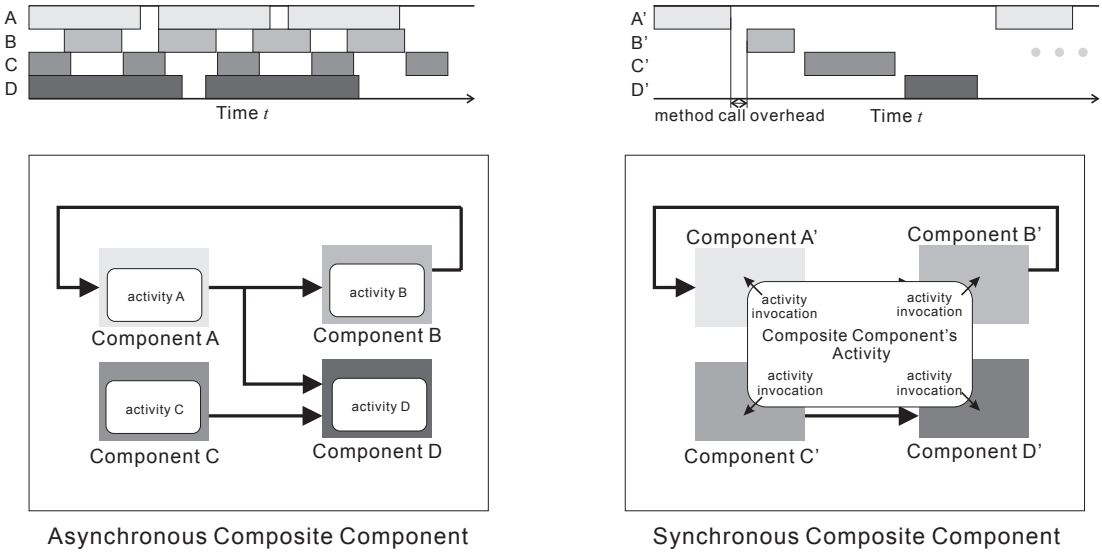

Fig. 7. Asynchronous Composite Component / Synchronous Composite Component. Asynchronous composit component provides components grouping and InPort/OutPort sharing functions. Synchronous composite component also provides them and serialized activity invocation. Its child components share their activity.

Table II shows experimental result. in all case, All components are executed in serial. From the mean execution-time and the standard deviation of the table II, it turns out that the deadline was kept in 1 ms. The call overhead per one component, which is estimated from total call overhead time, are from 2.6 to 4.8 ms. From these results, components' activity call overhead is very small and neglectable.

From this experiment, even if the component was implemented individually, two or more they can be combined if the components are implemented as RT-Components. Moreover, it was shown that these components were executed as if they were one real-time task.

B. Force controlled manipulator system

A synchronous composite component was applied to force controlled manipulator system. An end-effector force/torque sensor component, a manipulator component, a joystick component and a controller component are loaded into a synchronous composite component. The synchronous composite component was executed as a 2 ms periodic task in real-time.

As shown in figure 9,

1) End-effector force/torque sensor component,
2) Joystick component,
3) Controller component,
4) Manipulator component,

were executed in this order serially.

TABLE II

EVALUATION OF METHOD CALL OVERHEAD OF SYNCHRONOUS COMPOSITE COMPONENT. EACH COMPONENT'S ACTIVITY IS EMPTY.

Number of components	1	2	10
Mean execution time [ms]	1.00	1.00	1.00
Std. dev. [ns]	86.6	66.1	47.3
Total call overhead time [μs]	4.84	7.85	30.4
Call time per component			
Call overhead time [μs]	4.84	2.62	3.04
Std. dev. [ns]	86.6	121	92.1

All components are executed in real-time Linux (ARTLINUX) on PC (Pentium4 2.8GHz). Real-time period was set to 2.00 ms because of manipulator's motion controller board period.

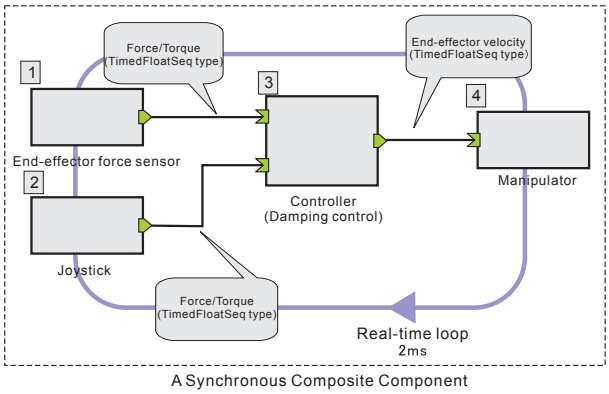

Fig. 8. Manipulator force control system by using Synchronous Composite Component: upper-left number means execution order.

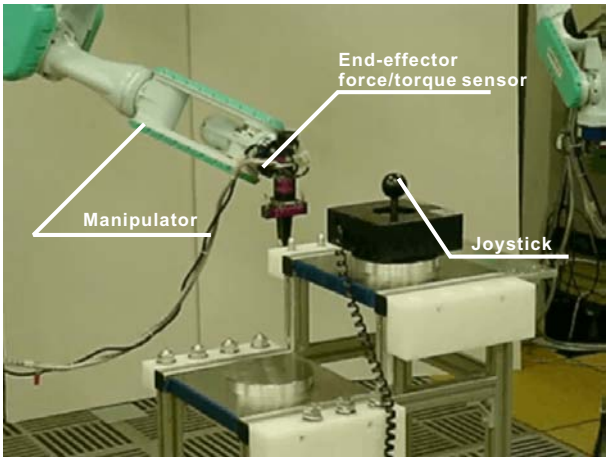

Fig. 9. Endeffector force srnsor, manipulator, joystick.

Table III shows task execution time statistics in this experiment. You can find the real-time task, which includes four components' activity, keeping 2 ms period. In the experiment, when force was applied to end-effector force/torque sensor, it was confirmed that an end-effector position moves in the direction of force. When force was applied to the joystick, it was confirmed that the end-effector position of the manipulator moves similarly. It was confirmed that force control is performed stably in these case.

TABLE III

EXECUTION TIME OF FORCE CONTROLLED MANIPULATOR SYSTEM.

Task period time	2.00 ms
Maximum execution time	2.01 ms
Minimum execution time	1.99 ms
Mean execution time	2.00 ms
Standard deviation	4.41 μs

The point is that these three devices components and one control component are not a monolithic program but programs completely created separately. Furthermore, it is important that these components were executed synchronously as a real-time task.

VI. CONCLUSION

In this paper, the necessity for the systematic methodology for constituting and integrating a robot system was discussed at first. Moreover, the necessity for a platform "RT-Middleware", which supports system integration in implementation level, was described.

New component framework "Composite Component" in RT-Middleware are introduced. It was shown that composite component framework, which realizes low level and real-time composition of independent RT-Components.

Finally, evaluation experiments were performed, and it was confirmed that the performance of the new framework, which executes some independent components in real-time, is high enough to be able to apply it to robot control.

In this paper, we focused attention on the RT-Middleware concepts and the RT-Component object model as main topic. To apply this framework to various real-time robotic systems and evaluating the performance of this framework will be performed in future work.

REFERENCES

[1] M.Mizukawa, H.Matsuka, T.Koyama, T.Inukai, A.Noda, H.Tezuka, Y.Noguchi, N.Otera, "ORiN Open robot Interface for the Network – The Standard Network Interface for Industrial robots and its Applications –", ISR2002, No.45

[2] Makoto Mizukawa, Hideo Matsuka' Toshihiko Koyama, Toshihiro Inukai, Akio Noda, Hirohisa Tezuka, Yasuhiko Noguchi, Nobuyuki Otera, "ORiN: Open Robot Interface for the Network – The Standard and Unified Network Interface for Industrial Robot Applications –", SICE Annual Conference 2002, pp.1160-1163, Osaka

[3] Orocos: Open Robot Control Software. http://www.orocos.org

[4] C. Schlegel, R. Worz, "The Software Framework SmartSoft for Implementing Sensorimotor Systems", IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '99, pp.1610-1616, Kyongju, Korea, October '99

[5] Fumio OZAKI, "Open Robot Controller Architecture (ORCA)", IROS2004 Workshop on Robot Middleware toward Standards,

[6] Fumio OZAKI, "Open robot controller archtecture (ORCA)", AIM2003 Workshop: Middleware Technology for Open Robot Architecture

[7] Kohtaro SABE, "Open-R : An Open Architecture for Robot Entertainment" , AIM2003 Workshop: Middleware Technology for Open Robot Architecture